20. Ratinho T., Harms R., Walsh S., Structuring the Technology Entrepreneurship publication landscape: Making sense out of chaos//Technological Forecasting and Social Change. 2015. vol. 100. pp. 168-175.

21. Scott S., VenkataramanS.The Promise of Entrepreneurship as a Field of Research//The Academy of Management Review. 2000. vol. 25. pp. 217–226.

22. Canadian Academy of Engineering. Wealth Through Technological Entrepreneurship// Engineering Issues. 1998. №7. pp. 1-2.

Старовойтов Иван Николаевич
ФГБОУ ВО «Курганский государственный университет»,
студент кафедры «Безопасность информационных и автоматизированных систем»,
starovoytovivan@gmail.com, Курган, Россия
Ревняков Евгений Николаевич
ФГБОУ ВО «Курганский государственный университет»,
канд. техн. наук, доцент кафедры «Безопасность информационных и автоматизированных систем»,
aphaline@mail.ru, Курган, Россия
Полякова Елена Николаевна
ФГБОУ ВО «Курганский государственный университет»,
канд. пед. наук, доцент кафедры «Безопасность информационных и автоматизированных систем»,
penelena1972@yanex.ru, Курган, Россия

ПАРАЛЛЕЛЬНЫЕ ВЫЧИСЛЕНИЯ НА ГРАФИЧЕСКИХ ПРОЦЕССОРАХ

УДК 004.272.44

Аннотация. На данный момент современные компьютеры обладают высоким уровнем производительности, позволяющим выполнять большое количество вычислений. Несмотря на то, что вычислительные мощности графических процессоров значительно опережают мощности центральных процессоров, CPU по-прежнему играет важнейшую роль в компьютере, а GPU подходит для решения только определенных задач или алгоритмов.
Ключевые слова: GPU, CPU, параллельные вычисления, CUDA, метод Монте-Карло

Abstract. At the moment, modern computers have a high level of performance, allowing you to perform a large number of calculations. Despite the fact that the computing power of GPUs is significantly higher than the power of central processors, the CPU still plays a crucial role in the computer, and the GPU is suitable for solving only certain tasks or algorithms.
Keywords: GPU, CPU, parallel calculations, CUDA, Monte Carlo method

Изначально центральные и графические процессоры были предназначены для разных задач. Центральные процессоры используются для создания обычных приложений, графические – для работы с трехмерной графикой и изображениями. В настоящее время становится все более популярным проведение вычислений с помощью графического процессора. Изначально все кажется очевидным: графические процессоры обладают намного большей производительностью чем центральные. Однако нельзя утверждать, что все вычисления можно перенести с центрального процессора на графический и для этого необходимо рассмотреть архитектуру CPU и GPU. Несмотря на некоторые общие моменты, графические и центральные процессоры имеют существенные различия. Список различий между CPU и GPU приведен в Таблице1.

Таблица 1 – Различия между CPU и GPU

Признак	CPU	GPU
Архитектура	Использует MIMD	Использует SIMD
Функции ядра	Каждое ядро работает отдельно от остальных, исполняя разные инструкции для разных процессов	Ядра выполняют одни и те же инструкции одновременно
Распараллеливание	Выполнение как можно большего количества инструкций одновременно	Выполнение всех необходимых операций независимо друг от друга
Доступ к памяти	Непредсказуемый	Связанный и легко предсказуемый
Кэш-память	До 16 Мб	48–64 Кб
Многопоточность	1–2 потока на ядро	До 1024 потоков на мультипроцессор
Устройство	Буферы команд, аппаратное предсказание ветвления и огромные объемы кэш-памяти	Исполнительные блоки

Можно сказать, что, в отличие от современных универсальных CPU, графические процессоры предназначены для параллельных вычислений с большим количеством арифметических операций. Выполнение расчетов на

GPU показывает отличные результаты в алгоритмах, использующих параллельную обработку данных, то есть, когда одну и ту же последовательность математических операций применяют к большому объему данных.

Для выполнения параллельных вычислений на графическом процессоре, разработчикам из компании NVIDIA была разработана универсальная архитектура параллельных вычислений – NVIDIA CUDA (Compute Unified Device Architecture). Она включает набор инструкций архитектуры ISA (Instruction Set Architecture) и реализацию параллельных вычислений на GPU. Для программирования разработчик может использовать наиболее распространенные языки высокого уровня C и C++ [1].

Архитектура CUDA разрабатывались с учетом двух основных целей:

– предоставить программисту функции и возможности, расширяющие стандартные языки программирования и позволяющие реализовывать только параллельные алгоритмы;

– предоставить поддержку для гетерогенных вычислений, то есть таких вычислений, в которых одновременно используется центральный и графический процессор таким образом, что последовательные части алгоритма выполняются на CPU, а параллельные – на GPU.

Перед тем, как рассматривать CUDA с программной точки зрения, необходимо рассмотреть такие понятия, как поток, варп, блок и сетка.

Поток (*thread*) – базовый набор данных для обработки. Стоит отметить, что поток CPU и GPU имеет разные значения. И как уже было отмечено ранее, переключение между двумя потоками не ресурсоемкая операция на GPU.

Варп (*warp*) – минимальный объем данных, обрабатываемых SIMD – способом в мультипроцессорах CUDA. Иным словами, это группа, состоящая из 32 потоков.

Блок (*block*) – группа связанных между собой потоков. Максимальное количество блоков составляет 65535.

Сеть (*grid*) – набор блоков, который должен быть обработан. Сеть может иметь трехмерную размерность.

С программной точки зрения, CUDA состоит из набора расширений к языку C или C++ и имеет следующие виды расширений:

1. расширения функций, которые показывают, чем вызывается и чем выполняется данная функция;

2. работа с памятью, включающая выделение, освобождение и копирование памяти;

3. предопределенные переменные, облегчающие работу с сеткой, блоками и потоками;

4. вспомогательные функции, позволяющие, например, считать время выполнения программы.

Для того, чтобы выяснить действительно ли GPU опережает CPU на параллельном выполнении арифметических операций, выполним тестирование программ, в которых происходит:

1. последовательное выполнение вычислений;
2. распараллеливание вычислений на CPU с помощью OpenMP;
3. выполнение вычислений на GPU с помощью NVIDIA CUDA.

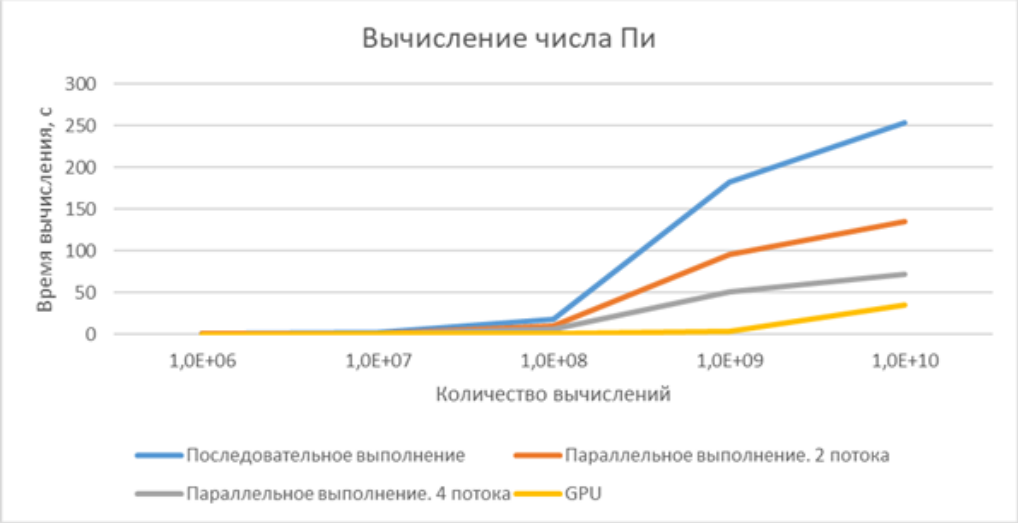
В качестве тестовой задачи возьмем вычисление числа π методом Монте-Карло. В данном методе используется четверть круга единичного радиуса, заключенного в единичный квадрат. Суть метода состоит в случайном разбросе точек по площади квадрата и расчете числа π в зависимости от количества точек [4].

Для тестирования использовался процессор Intel Core i5-7500 с характеристиками техпроцесс – 14 нм, количество ядер – 4, количество потоков – 4, базовая тактовая частота – 3.4 ГГц.

Для тестирования использовалась видеокарта GeForce GTX 1060 с характеристиками память – 6 Гб GDDR5/X, количество CUDA-ядер – 1280, техпроцесс – 16 нм, базовая тактовая частота – 1506 МГц.

Для параллельных вычислений на CPU, мы используем стандарт параллельного программирования OpenMP, который позволяет разбивать выполнение программы на потоки. Для тестирования используется 2 и 4 потока. Дальнейшее увеличение числа потоков не приведет к ускорению [2].

Результаты тестирования выглядят следующим образом.



Результаты тестирования

Из результатов тестирования можно увидеть, что проведение вычислений на GPU дает ускорение более чем в 7 раз по сравнению с последовательным вычислением на CPU и более чем в 2 раза по сравнению с параллельным вычислением на CPU.

На основании проведенного тестирования можно сделать следующие выводы:

- использование GPU для вычислений приводит к значительному росту производительности, однако этот рост полностью связан с программой, а именно с кодом, который можно выполнять параллельно. В иных случаях ускорение может быть другим, или вовсе отсутствовать. Следовательно, важно умение программиста преобразовывать код программы по параллельное выполнение;

- центральный процессор остается загруженным, а значит, применение мощных, многоядерных центральных процессоров в системах остается актуальным.

Список использованной литературы

1. CUDA C++ Programming Guide. [Электронный ресурс] // Документация по CUDA Toolkit. URL: https://docs.nvidia.com/cuda/cuda-c-programming-guide/index.html (дата обращения: 24.06.2020)

2. Директивы OpenMP. [Электронный ресурс] // Документация Microsoft для конечных пользователей, разработчиков и ИТ-специалистов. URL: https://docs.microsoft.com/ru-ru/cpp/parallel/openmp/reference/openmp-directives?view=vs-2019 (дата обращения: 24.06.2020)

3. Конструирование и оптимизация параллельных программ. Новосибирск: Ин-т систем информатики имени А. П. Ершова СО РАН, 2008. 332 с.

4. Метод Монте-Карло. [Электронный ресурс] // Википедия. URL: https://ru.wikipedia.org/?oldid=106074536 (дата обращения: 24.06.2020)

Портнов Михаил Семенович
кандидат социологических наук, доцент.
Чебоксарский кооперативный институт (филиал)
Российского университета кооперации, доцент.
E-mail: m.s.portnov@rucoop.ru, г. Чебоксары, Россия.
Речнов Алексей Владимирович
кандидат педагогических наук, доцент. Чебоксарский
кооперативный институт (филиал) Российского
университета кооперации, доцент.
E-mail: a.v.rechov@rucoop.ru, г. Чебоксары, Россия.
Филиппов Владимир Петрович
кандидат физико-математических наук, доцент.
Чебоксарский кооперативный институт (филиал)
Российского университета кооперации, доцент.
E-mail: v.p.filippov@rucoop.ru, г. Чебоксары, Россия.

ТЕХНОЛОГИЯ ПРЕПОДАВАНИЯ КУРСА «КОМПЬЮТЕРНЫЕ СЕТИ»: ПРАКТИЧЕСКИЙ ПОДХОД